jamk.fi

Rakennustarkastuksien dokumentointi web-sovelluksena

MEAN-sovelluskokoelma

Jaakko Seppänen

Opinnäytetyö
Syyskuu 2015
Ohjelmistotekniikan koulutusohjelma
ICT-ala

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Seppänen, Jaakko	Opinnäytetyö	23.3.2016
	Sivumäärä 38	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Rakennustarkastuksien dokumentointi web-sovelluksena
MEAN-sovelluskokoelma

Koulutusohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Salmikangas, Esa

Toimeksiantaja(t)
PJR-Team Oy

Tiivistelmä

Opinnäytetyönä tehtiin web-sovellus, jonka tarkoituksena on helpottaa ja nopeuttaa rakennus-/kosteustarkastuksien dokumentointia pdf-muotoon. Lisäksi sovellus toimii tietokannan hallintasovelluksena. Työn toimeksiantajana toimi rakennusfirma PJR-Team Oy, joka on erikoistunut rakentamisen ohjaus-, valvonta- ja työnjohtopalveluihin.

Tämä sovellus on osa kaksiosaista järjestelmää, jossa toinen osa on rinnalla kehitetty android-sovellus. Android-sovelluksen tarkoitus on helpottaa tietojen keräämistä tarkastuksia tehdessä ja synkronoida nämä tiedot yrityksen tietokantaan. Järjestelmän toinen osa, web-sovellus, kerää nämä tiedot kyseisestä tietokannasta hallinnointia varten. Jokainen tarkastus on mahdollista tulostaa pdf-tiedostona.

Tämän opinnäytetyön toteutusosan web-sovellus tehtiin hyödyntämällä MEAN-sovelluskokoelmaa, joka koostuu MongoDB:stä, Express.js:stä, Angular.js:stä ja Node.js:stä. Sovelluskokoelma valittiin, koska kokemusta oli aikaisemmin Angular.js:stä ja tarkempaa tutkimusta tehdessä huomattiin, että tämä kokoelma toimii erittäin hyvin keskenään.

Lopputuloksena saatiin aikaiseksi toimiva web-sovellus hieman puutteellisina toiminnallisuuksina, mutta nämä ovat helposti lisättävissä. Koko järjestelmää ei ole testattu oikeassa käytössä, koska tämän opinnäytetyön valmistuessa android-sovellus –osaa ei ole vielä kehitetty loppuun asti.

Opinnäytetyö käy läpi, kuinka tällainen web-sovellus suunnitellaan ja kuinka se toteutetaan käyttämällä MEAN-sovelluskokoelmaa. Se myös selittää, miten jokainen teknologia toimii ja miksi niitä käytetään. Lopussa selvitetään ja pohditaan myös projektin tuloksia.

Avainsanat (asiasanat)
Web-sovellus, Suunnittelu, MEAN-sovelluskokoelma, Tietokanta, JavaScript, AngularJS

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Seppänen, Jaakko	Bachelor's thesis	23.3.2016
		Language of publication: Finnish
	Number of pages: 38	Permission for web publication: x

Title of publication
Building inspection documentation web application
MEAN-stack

Degree programme
Software Engineering

Tutor(s)
Salmikangas, Esa

Assigned by
PJR-Team Oy

Abstract

The purpose of this thesis is to create a web-application that eases inspections done to buildings by making the procedure faster to document into pdf-format. In addition, this application works as a management tool for a database used by this system. The client of this project is a construction company PJR-Team Oy, who specializes in guiding and inspecting building processes.

This web-application is part of a two-part system where the other part is an android-application. The android-application is supposed to ease the collection of data during construction inspections and synchronize this data to the company's database. The second part of the system is a web-application which collects this data from the database for management. Every inspection is supposed to be able to be printed out as a pdf-format.

The web-application in which this thesis focuses on uses a MEAN-stack, which is a collection of several technologies: MongoDB, Express.js, Angular.js and Node.js. MEAN-stack was selected because it uses Angular.js and the team working on this project already has some experience in it. Also during further investigation, it was discovered that this stack of technologies fills every purpose this project needs.

The result was a functional web-application with a few missing requirements due to time restrictions. The whole system was never tested, because the android-part was never developed during this thesis.

This thesis gives insight on how to make a web-application that uses a database by using a MEAN-stack and it also explains what each of the technologies used in this project actually do and how they work. In the end the project results are also explained and analyzed.

Keywords/tags (subjects)
Web application, Planning, MEAN-stack, Database, JavaScript, AngularJS

Miscellaneous

SISÄLTÖ

KUVIOT

SANASTO

Agile

Agile software development (suom. ketterä ohjelmistokehitys) on ohjelmistotuotannon läpiviennissä käytettäviä menetelmiä.

Angular.js

Angular.js (tai AngularJS) on JavaScript-viitekehys. Angular.js on yksi neljästä teknologiasta MEAN-sovelluskokoelmassa.

Debug

Debuggaus (suom. virheenkorjaus) on testauksen aikana löytyneiden virheiden paikantaminen ja niiden korjaaminen.

Express.js

Express.js on palvelin-viitekehys Node.js:ään pohjautuviin web-sovelluksiin.

Git

Git on avoimeen lähdekoodiin perustuva versionhallintaohjelmisto.

GWT

GWT eli Google Web Toolkit on avoimeen lähdekoodiin perustuva kehitystyökalu monitahoisiin web-sovelluksiin. Se tarjoaa monia pienoisohjelmia joilla voi ohjelmoida sovelluksia Java-ohjelmointikielellä, mikä käännetään JavaScriptiksi.

Java

Java-ohjelmointikieli on Sun Microsystemsin kehittämä laitteistoriippumaton oliopohjainen ohjelmointikieli.

JavaScript

JavaScript on Web-ympäristössä käytettävä oliopohjainen skriptikieli, millä automatisoidaan ohjelma ilman, että tarvittaisiin laajaa ohjelmointikieltä.

JSON

JSON eli JavaScript Object Notation on avoimen standardin tiedostomuoto, mitä käytetään asynkroniseen tiedostonvälitykseen palvelimen ja selaimen välillä.

MEAN-stack

MEAN-stack (vapaasti suomennettuna MEAN-sovelluskokoelma) on avoimeen lähdekoodiin perustuva JavaScript sovelluskokoelma, mitä käytetään web-sovellusten ohjelmointiin.

MongoDB

MongoDB on JSON-dokumentteihin pohjautuva tietokanta. Tätä tyyliä kutsutaan myös NoSQL-pohjaiseksi tietokannaksi.

Mongoose

Mongoose tarjoaa mahdollisuuden simuloida web-sovelluksen dataa tavallisten tietokantamallien mukaisesti.

MVC

MVC eli model-view-controller (suom. malli-näkymä-käsittelijä) on ohjelmistoarkkitehtuurityyli, mitä esimerkiksi Angular.js käyttää.

Node.js

Node.js on avoimeen lähdekoodiin perustuva web-sovelluksissa käytetty ajoympäristö palvelinpuolella.

Repositorio

Repositorio on paikka, mihin sovelluspaketit ovat säilötty. Sovelluspaketit voi hakea ja asentaa repositoriosta tietokoneelle.

REST

Representational State Transfer on ohjelmistoarkkitehtuurityyli web-pohjaisiin sovelluksiin, mikä käyttää http-protokollaa kommunikointiin palvelimen ja sovelluksen välillä.

Rikkaat Internet-sovellukset

Internet-sovellukset, joiden ominaisuus ja toiminnallisuus muistuttaa työpöytäsovelluksia.

routeProvider

routeProvider on Angular.js:n palvelu, mikä käsittelee selaimen url:it ja kartoittaa ne sovelluksessa oikeille html-sivuille ja kontrollereille.

Scrum

Ketterässä ohjelmistokehyksessä käytetty projektihallinnan viitekehys.

1 OPINNÄYTETYÖN LÄHTÖKOHDAT

1.1 TAUSTA

Tämän opinnäytetyön toimeksiantajana toimii yhtiö PJR-Team Oy, joka tekee mm. rakennustarkastuksia ja kosteusmittauksia erilaisiin kohteisiin. Nämä tarkastukset suoritetaan hyvin manuaalisesti: tiedot ja kommentit tarkastuksista kirjoitetaan käsin paperille ja tulokset mittaristoista tallennetaan usein ottamalla kuva niistä. Myöhemmin nämä tiedot kirjoitetaan tietokoneella pdf-dokumentiksi ja arkistoidaan.

Tämä käsin tehty prosessi ei ole kuitenkaan hidas, mutta tietojen kirjoittaminen talteen useampaan kertaan ei ole ajallisesti tehokasta. Lisäksi tietojen ja mittaustulosten arkistointi ja muokkaaminen olisi helpompaa, jos ne kaikki olisivat tietokannassa tallessa. Näin heräsi idea: voiko tarkastus- ja mittausprosesseja nopeuttaa ja helpottaa erillisen ohjelman avulla?

1.2 TYÖN TAVOITTEET JA RAJAUKSET

Ongelma on haastava ja tämänkaltaiset ohjelmat tai järjestelmät ovat äärimmäisen harvassa, etenkin Suomessa. Tarkastusprosessi itsessään on jo nopea, ja työntekijät ovat oppineet toimimaan sen kanssa tehokkaasti. Tämän ympärille rakennettava järjestelmän tulisi siis olla helppo ja yksinkertainen käyttää. Koska PJR-Team Oy ei ollut mikään valtava yhtiö tätä opinnäytetyötä tehdessä, oli se erinomainen kohde testata tällaista järjestelmää.

Tarvittava järjestelmä tarkastuksille on helppo jakaa kahteen osaan. Ensimmäisessä osassa pohditaan ratkaisu siihen, miten kerätään tieto jo tarkastuksia tehdessä. Toisessa osassa tämä tieto pitäisi olla helposti muunnettavissa pdf-dokumentiksi automaattisesti

”kentältä” kerätyistä tiedoista. Tästä päädyttiin tekemällä järjestelmästäkin kaksiosainen: mobiiliosa, jolla kerätään tiedot ja tallennetaan tietokantaan ja web-sovellusosa, jolla voi helposti tarkastella tietokannan tietoja ja lisätä/muokata/poistaa tarvittavia tietoja tietokannasta. Myös pdf-dokumentin luonti tapahtuu web-sovelluksella. Lisäksi tietokannan hallinta on tarvittaessa helppo suojata yksinkertaisella kirjautumisjärjestelmällä, mikä suojaa asiakastietoja.

1.3 RAKENNUSFIRMA PJR-TEAM OY

Toimeksiantajana toimii rakennusfirma PJR-Team Oy, joka on erikoistunut rakentamisen ohjaus-, valvonta- ja työnjohtopalveluihin. Näiden lisäksi PJR-Team Oy tekee rakennustarkastuksia ja kosteusmittauksia. (PJR-Team N.d.)

PJR-Team Oy:n yhteyshenkilönä toimi Jorma Heiskanen, joka toimi projektin tilaajana ja teki vaatimusmäärittelyn yhdessä projektin muiden jäsenten kanssa. Hän oli myös yhteyshenkilö, jos lisätietoja projektikehityksessä tarvittiin.

2 SOVELLUKSEN SUUNNITTELU

2.1 WEB-SOVELLUS

2.1.1 VAATIMUKSET

Opinnäytetyön vaatimukset saatiin selville haastattelemalla PJR-Teamin työntekijää Jorma Heiskasta. Haastattelun aikana luotiin muistiinpanot funktionaalisista ja ei-funktionaalisista vaatimuksista ja samalla ne lajiteltiin tärkeysjärjestykseen. Näin oli

helppo huomata, mitkä ominaisuudet jäivät niin sanottuun ”nice to have” –kategoriaan aikataululaskentaa tehdessä. Vaatimukset kirjattiin Confluence-nimiseen wikiohjelmistoon.

Tietojen salaamisesta ei tehty mitään vaatimuksia, mutta päätettiin, että ne olisi hyvä suojata yksinkertaisella kirjautumissysteemillä.

2.1.2 Aikataulutus

Aikataulutus suunniteltiin projektijäsenten kesken. Jokaiseen tehtävään kuluva tuntimäärä arvioitiin ja kirjattiin wikiohjelmistoon, josta ne voidaan myöhemmin lisätä tehtävienhallintaohjelmistoon. Näin saatiin hyvä käsitys siitä, mitä ominaisuuksia pystytään aikamääreessä toteuttamaan, ja mitkä jäävät mahdollisesti kehittämättä. Valitettavasti tähän aikatauluun ei otettu huomioon tarpeeksi hyvin teknologioiden opettelemiseen kuluva aika.

2.1.3 Projektinhallinta

” Yksittäisten tehtävien hallinta on avain projektien onnistumiseen: tehtävänhallinnan avulla huolehditaan, että osatehtävät tulevat suoritetuksi ja kokonaisuus valmistuu. ” (Jira (ohjelmisto) 2015).

Projektin alkuperäisenä ideana oli kehittää molemmat osat järjestelmää (mobiili- ja web-sovellus) samanaikaisesti kahden eri opiskelijan toimesta. Tehtävien kirjaus ja niiden aikataulutus tapahtui JIRA-nimisellä tehtävienhallintaohjelmalla. Näin varmistettiin, etteivät tehtävät unohdu. Lisäksi JIRA näyttää graafisesti projektin etenemisen mm. burndown-kaaviolla, mistä näkee, kuinka lähellä ollaan suunniteltua edistymistä ja paljonko työtä on tehty tunneissa missäkin projektin vaiheessa. Muistiinpanot ja suunnitelmat kirjattiin Confluence-nimiseen organisaatiowikiohjelmistoon, mikä toimii yhdessä JIRA:n kanssa.

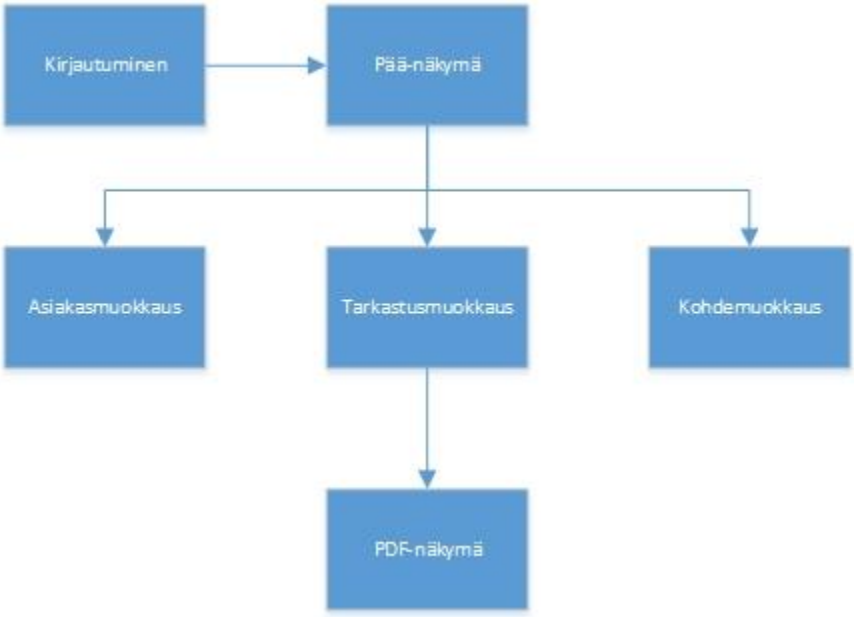
Projektihallinnan viitekehyksenä valittiin scrum, koska se on todettu tehokkaaksi ketterässä ohjelmistokehityksessä. Lisäksi projektin jäsenillä oli jo ennestään kokemusta scrumista ja sen toiminnasta. Projekti jaettiin 7 eri kehitysjaksoon (sprinttiin), joista jokainen kesti noin 2 viikkoa.

Versiohallinta toteutettiin Gitillä, koska se mahdollistaa koko versiohistorian käsittelyn paikallisesti ilman keskitettyä palvelinta ja helpon yhdistämisen useiden repositorioiden välillä.

2.1.4 Sovelluksen rakenne

Sovellus koostuu front end ja back end -osioista. Front end kattaa asiakasohjelman, eli minkä kanssa asiakas vuorovaikuttaa. Back end kattaa palvelinpään, mikä sisältää mm. tietokannan ja muun teknologian, minkä kanssa asiakas ei suoraan toimi.

Asiakasohjelma koostuu eri näkymistä, mitä käyttäjä pystyy navigoimaan. Alla olevassa kuviossa (Kuvio 1) on selvitetty, miten näkymistä voidaan navigoida toisiin näkymiin. Esimerkkinä kuinka tarkastusmuokkaus-näkymästä siirrytään PDF-näkymään. Nuolet toimivat myös päinvastoin, jos näkymästä halutaan palata takaisin. Asiakasohjelman avatessa aloitetaan aina kirjautumisnäkymästä, jos edellisestä istunnosta on kirjauduttu ulos.



KUVIO 1, ASIAKASOHJELMAN NÄKYMÄT

2.2 TIETOKANTA

2.2.1 TIETOKANNAN SUUNNITTELU

Tietokannan suunnittelu alkaa vaatimusmäärittelyn jälkeen. Kun vaatimukset ovat tiedossa, on helppo sopia, mitkä tiedot tulee tietokantaan tallentaa (käsiteanalyysi). Suurin haaste oli löytää hyvä ratkaisu, kuinka data tallennetaan mobiililaitteen ja web-applikaation välillä. Lähtökohdaksi päätettiin valita NoSQL-pohjainen tietokanta MongoDB web-applikaatiolle, koska se on joustava ja antaa paljon mahdollisuuksia lähestyä kyseistä ongelmaa. Lisäksi se nopeuttaa suunnitteluvaihetta, koska NoSQL-tietokantaan ei tarvitse erikseen tehdä tietokantamallia. Kaikki siihen liittyvä kehitys tapahtuu siis itse koodissa ja tämä oli tärkein syy sen valitsemiseen.

2.2.2 KÄSITEANALYYSI

” Tarkoituksena on saada aikaan *riittävän tarkka* kuva tietosisällöstä, jotta tätä voisi jatkossa käyttää pohjana tietojärjestelmien toimintojen määrittelylle ja toisaalta lähtökohtana tietokannan tai muun tarkasteltavan tietokokoelman, esimerkiksi viestiliikenteen, rakennetason suunnittelulle. Analyysiä joudutaan suorittamaan myös tilanteissa, joissa sovitetaan yhteen eri tietokantoja tai tietokokoelmia. ” (Laine, H. 2005).

Vaikka kyse on NoSQL-tietokannasta, jonkinnäköinen malli kannattaa visualisoida. Taulujen suhteet selvitetään etukäteen käsiteanalyysillä, jotta ei ohjelmointivaiheessa tule epäselvyyksiä. Toinen tärkeä syy taulujen suhteiden selvittämiseen on, että projektin mobiiliosan on tarkoituksena käyttää SQL-pohjaista tietokantaa (MySQL) ja sen kääntäminen NoSQL:ksi on helpompaa, jos suhteet tauluilla ovat pääosin identtisiä.

2.3 REST-RAJAPINTA

Tiedot tulee jollain tavalla siirtää tietokannasta asiakaspääteohjelmaan käyttäjälle muokattavaksi. Sovelluksessa tämä tapahtuu pyytämällä haluttu tieto tietokannasta REST-kutsulla, jolloin tieto muunnetaan JSON-muotoon. Tässä muodossa sitä on helppo käsitellä ja se pystytään myös lähettämään takaisin tietokantaan muutettuna tai täysin uutena. Node.js hoitaa yhdessä Mongoosen kanssa, että tieto haetaan tietokannasta ja tallennetaan takaisin tietokantaan oikein.

REST-rajapinta on rajapinta palvelimen (server) ja asiakasohjelman (client) välillä. REST käyttää HTTP-metodeja eksplisiittisesti. Jos palvelimelta halutaan dataa, käytetään komentoa GET. Jos taas halutaan luoda uutta dataa palvelimelle, käytetään komentoa POST.

REST on myös tilaton. Tämä tarkoittaa, että jos jonkin osasen tila pitää sovelluksessa tallentaa (esimerkiksi millä sivulla käyttäjä milläkin hetkellä on), ei tieto tästä tallennu palvelinpäähän. Sen sijaan tilojen tallentaminen tapahtuu asiakkaan päässä kekseillä

(cookies) tai muilla tavoilla. Angular.js on erittäin hyvä tässä, sillä se tarjoaa paljon ominaisuuksia asiakasohjelman päähän.

Kolmas tärkeä ominaisuus REST:llä on, että se pystyy siirtämään dataa palvelimen ja asiakasohjelman välillä JSON-muodossa. Tämä tekee datan käsittelystä JavaScriptillä huomattavasti helpompaa. JSON ei myöskään rasita palvelinta niin paljon, koska uutta dataa ei tarvitse erikseen palvelimella käsitellä. Se vain haetaan sieltä asiakasohjelmaan, jossa itse käsittely tapahtuu. (Buecheler, C. 2015.)

3 TYÖKALUT JA TEKNOLOGIA

3.1 KEHITYSYMPÄRISTÖ WEBSTORM

Web-sovellus kehitettiin WebStorm-nimisellä kehitysympäristöllä, mikä toimii erinomaisesti MEAN-sovelluskokoelman kanssa. WebStorm on kevyt ja hyvin varusteltu asiakasohjelmien kehittämiseen Angular.js:llä sekä palvelinpuolen kehittämiseen Node.js:llä. Jotta WebStorm toimisi kätevästi MongoDB:n kanssa, täytyy siihen asentaa "Mongo Plugin” -lisäosa. Tämän avulla WebStormin voi yhdistää käynnissä olevaan MongoDB-tietokantaan.

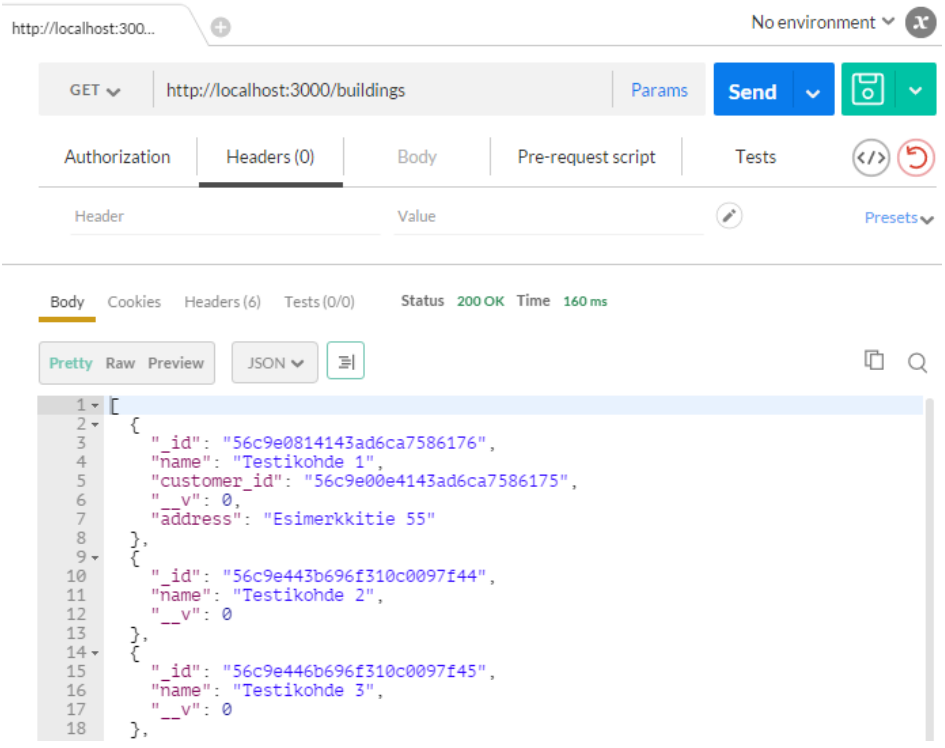
3.2 JULKINEN ARKISTO NPM

Npm on julkinen paketinhallintatyökalu tarkoitettu Node.js:lle. Se auttaa JavaScript-kehittäjiä helpottamalla koodin jakoa, uudelleenkäyttöä ja jo valmiiksi jaetun koodin päivittämistä. (What is npm? n.d.). Npm:llä pääasiallisesti asennetaan tarvittavia Node.js moduuleita, jotka sisällytetään koodissa komennolla *require(’moduulin nimi’);*. Tässä projektissa käytettiin mm. seuraavia moduuleita: express, path, morgan, cookie-parser, body-parser ja mongoose. Näistä tärkeimmät ovat express ja mongoose. Npm tekee

kehittämisestä paljon nopeampaa, koska jokainen moduuli asennetaan aina yhdellä komennolla konsolista *npm install <Moduulin nimi>*. Tämän komennon jälkeen moduuli on automaattisesti asennettu projektiin ja se on valmis käytettäväksi.

3.3 POSTMAN

Postman on Google Chromeen asennettava lisäosa. Sillä voi luoda ja lähettää http-pyyntöjä palvelimelle ja Postmania usein käytetäänkin HTTP-pyyntöjen testaamiseen. Tämän sovelluksen kehittämisessä sitä käytetään REST-rajapinnan testaukseen. Postmanilla saa helposti selville, mitä http-pyynnöt tekevät ilman, että joutuisi erikseen kirjoittamaan koodia testausta varten (ks. Kuvio 2). Postman myös tallentaa kaikki tehdyt pyynnöt, jotta samojen asioiden testaus nopeutuisi.



KUVIO 2, KUVANKAAPPAUS POSTMANISTA

3.4 JIRA

JIRA on Atlassian-ohjelmistoyrityksen valmistama tehtävienhallintaohjelmisto. JIRA:a käytetään mm. projektienhallintaan ja erilaisten työmääräysten, virheiden ja tukipyyntöjen raportointiin. JIRA:n avulla suuretkin tehtävämäärät pysyvät järjestyksessä. Tehtävien suorittamisesta ja ongelmien ratkaisuista jää JIRA:an talteen tieto, joka löytyy helposti samanlaisen tilanteen toistuessa. (Jira (ohjelmisto). 2016.)

JIRA on kaupallinen ohjelmistotuote, jonka hinnoittelu perustuu maksimikäyttäjämäärään. Koska koko järjestelmää kehittää vain kaksi henkilöä, jää JIRA:n vuotuinen hinta hyvin pieneksi. Kymmenen tai vähemmän käyttäjämäärän hinta on vain 10 dollaria vuodessa. Jos henkilömäärä on taasen yli kymmenen, seuraava paketti maksaakin jo 1800 dollaria ensimmäiseltä vuodelta. JIRA:n voi ostaa joko omalle palvelimelle tai pilvipalveluna. (JIRA Software 2016).

3.5 CONFLUENCE

Confluence on Atlassian-ohjelmistoyrityksen valmistama organisaatiowikiohjelmisto. Se on verkkosivusto johon kaikki luvan saaneet voivat lisätä ja muokata projektiin liittyvää dokumentaatiota. Tämä dokumentaatio voi sisältää mm. tuotteen vaatimukset, projektisopimus, palaverimuistiinpanot, aikataulusuunnitelmat ja tehtävät. Koska Confluence on suunniteltu siten, että kaikki projektin dokumentit voi pitää tallessa yhdessä wiki-sivustossa, tekee se siitä hyvin suositun (ks. Kuvio 3). Tämä vähentää hämmennystä, jos dokumentit olisivat ovat tallennettu eri paikkoihin ja tekee projektin organisoinnista huomattavasti nopeampaa.

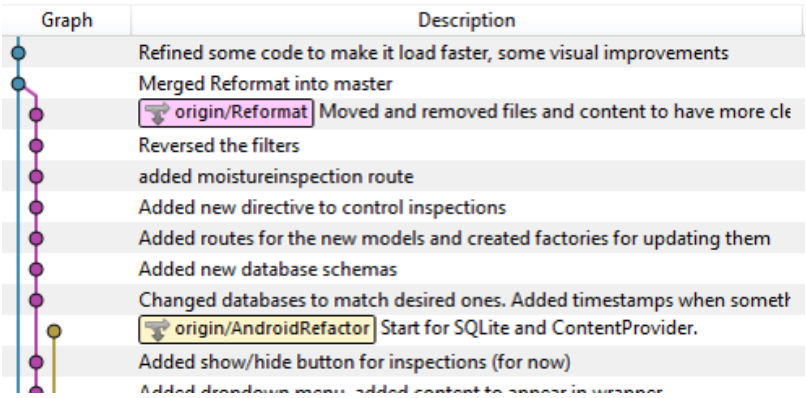
KUVIO 3, ATLASSIANIN NÄKEMYS KESKITETYSTÄ DOKUMENTOINNISTA (CONFLUENCE 2015)

Confluence toimii yhdessä JIRA:n kanssa ja siksi sieltä voi helposti kääntää esimerkiksi tuotevaatimukset suoraan tehtäviksi JIRA:an.

3.6 SOURCETREE JA BITBUCKET

SourceTree on kolmas tässä projektissa käytetty Atlassian – ohjelmistoyrityksen tuote. Se on graafinen Git ja Mercurial käyttöliittymä Windowsille tai Macille. SourceTree on ystävällinen työkalu niille, jotka eivät ole käyttäneet paljoa Git:iä tai eivät tykkää operoida sitä komentoriviltä. Se on selkeä ja yksinkertainen, mutta tarjoaa myös kaikki tarvittavat ominaisuudet asiantunteville. (SourceTree 2015).

SourceTree valittiin tähän projektiin sen takia, että se toimii yhdessä Bitbucketin kanssa. Sillä on myös helppo havainnollistaa eri samaan aikaan kehitettävät versiot projektista (ks. kuvio 4).



KUVIO 4, ESIMERKKI ERI VERSIOIDEN GRAAFISESTA NÄKYMÄSTÄ SOURCETREE:LLÄ

Bitbucket on myös Atlassianin omistama verkkopohjainen palvelu, joka käyttää Git – versiointihallintaohjelmistoa (vuodesta 2011 lähtien). (Bitbucket 2016).

3.7 TEKNOLOGIAT

Koko web-sovellus koostuu neljästä pääteknologiasta, joihin tässäkin opinnäytetyössä keskitytään: MongoDB, Express.js, Angular.js ja Node.js. Nämä neljä eivät kuitenkaan yksin riitä toteuttamaan kaikkia ominaisuuksia, mitä sovellukselta vaaditaan. Seuraavat kappaleet käsittelevät teknologioita, jotka liittyvät tavalla tai toisella MEAN-sovelluskokoelmaan tai ovat muulla tavalla tärkeitä tässä projektissa.

3.1.1 JAVASCRIPT

JavaScript on kevyt prototyyppi-pohjainen ja dynaaminen ohjelmointikieli, mikä tukee olioihin- ja funktioihin perustuvaa ohjelmointityyliä. Sillä yleensä ohjelmoidaan toiminnot web-sivustoihin, mutta sitä käytetään myös muissakin, kuin selainpohjaisissa ympäristöissä. Se on helppo oppia ja se muistuttaa syntaksiltaan Java- ja C++-ohjelmointikieliä. JavaScript ei kuitenkaan prototyyppi-pohjaisena kielenä ole missään tekemisissä tyypillisen Java-ohjelmointikielen kanssa.

Kolme pääteknologiaa MEAN-sovelluskokoelmassa perustuu JavaScriptiin ja siksi niillä on pääte .js. JavaScript on siksi tärkein teknologia tässä projektissa, vaikka suurin osa itse kirjoitetusta koodista ei olekaan puhtaasti JavaScriptiä. (What is JavaScript? 2015).

3.1.2 jQuery – kirjasto

jQuery on nopea, pienikokoinen ja paljon ominaisuuksia omaava JavaScript-kirjasto. Se tekee html-dokumentin manipulaatiosta, tapahtumien käsittelystä ja animaatioista paljon yksinkertaisempaa. jQuery tomii kaikilla nykypäivän suosituimmilla selaimilla. (What is jQuery? N.d.).

3.1.3 Bootstrap

Bootstrap on yksi suosituimmista html, CSS ja JavaScript viitekehyksistä responsiivisia verkkosivuja kehittäessä. Responsiivisuudella tarkoitetaan sitä, että verkkosivu muuttaa ulkonäköään automaattisesti verkkosivun leveyden vaihtuessa. Näin se pystytään ohjelmoimaan helposti myös mobiililaitteiden näytöille sopivaksi.

Tässä projektissa Bootstrapia ei kuitenkaan käytetty sen responsiivisien ominaisuuksien vuoksi, koska mobiililaitteelle kehitetään oma versio erikseen. Bootstrap valittiin sen takia, koska sillä on helppo kehittää siistin näköisiä verkkosivuja hyödyntämällä sen valmista CSS-kirjastoa ja valmiiksi tyyliteltyjä komponentteja.

3.1.4 Mongoose

Mongoose toimii tietokannan ja asiakasohjelman välillä. Sen tarkoitus on muuntaa oliot tietokannasta JavaScript-olioiksi ja pitää kevyttä rakennetta yllä tietokantamallista. Tällainen tietokantamallin simulaatio vähentää tietokannan joustavuutta, mutta

helpottaa myöhemmin datan kääntöä SQL:stä. Olioiden muunto toimii myös käsi kädessä Node.js:n kanssa, koska molemmat käyttävät JavaScriptiä ja JSON:ia.

4 MEAN-SOVELLUSKOKOELMA

4.1 YLEISTÄ

MEAN-sovelluskokoelma koostuu neljästä eri teknologiasta: MongoDB, Express.js, Angular.js ja Node.js, joiden alkukirjaimista sovelluskokoelman nimi muodostuu. Koska tarkoituksena oli tehdä puhdas web-applikaatio tietokannan kanssa, soveltui tämä kokoelma projektiin täydellisesti. MEAN-sovelluskokoelman kaikki osaset tukevat JavaScriptiä, mikä on pääasiallinen ohjelmointikieli web-applikaatioissa.

MEAN-sovelluskokoelman muina huomattavina etuuksina ovat sen joustavuus ja performanssi. Koska kokoelma tukee kokonaisuudessaan JavaScriptiä, on data tietokannassa saman näköinen, kuin asiakassovelluksessakin (JSON). Tämä nopeuttaa tietokantahallintaa ja virheiden etsintää (debug). (Karpov, V. 2016.)

Toinen tapa toteuttaa web-applikaatio olisi ollut puhtaasti GWT:llä, jossa Java-ohjelmointikieli käännettäisiin JavaScriptiksi ja HTML:ksi. Tämä toteutustapa kuitenkin hylättiin nopeasti MEAN-sovelluskokoelman ylivoimaisen kehittämisnopeuden ja -helppouden vuoksi.

4.2 MONGODB

MongoDB on NoSQL tietokantamalli, joka tallentaa dataa käyttäen joustavaa dokumentti-datamallia, mikä muistuttaa JSON:ia. Nämä dokumentit voivat sisältää

yhden tai useamman kentän, joissa voi olla taulukkoja, binääridataa tai jopa alidokumentteja. Joustavuus tulee siitä, että nämä kentät voivat vaihdella dokumentista riippuen. Tämä joustavuus mahdollistaa nopean datamallien muuttamisen, jos sovelluksen vaatimukset vaihtuvat, edistäen projektia tekemällä siitä mm. joustavamman ja ketterämmän kehittää. (MongoDB 2015).

Toisinkuin useimmat NoSQL-tietokannat, MongoDB tarjoaa kattavan toisarvoisen indeksoinnin, jonka ansiosta tietokantahaut eivät rajoitu pelkästään primääriavaimeen ja onnistuu myös mm. tekstihakuna. MongoDB tarjoaa myös muita ominaisuuksia, mitä yritykset tarvitsevat nykypäivän ohjelmistoissa, kuten laajan turvallisuuspalvelun ja kykenevyyden koosteisiin. (MongoDB 2015).

4.3 EXPRESS.JS

Express.js on hyvin pienikokoinen ja joustava node.js:n pohjalla toimiva web-sovelluksen viitekehys. Express.js tarjoaa paljon ominaisuuksia yksisivuisia web-sovelluksia tehdessä, mutta soveltuu myös monisivuisien web-sovelluksien kehittämiseen. Vaikka Express.js itsessään on hyvin minimaalinen, voi siihen asentaa paljon ominaisuuksia liitännäisillä (engl. plugin). Express.js on back end-osa MEAN-sovelluskokoelmassa yhdessä MongoDB:n ja Angular.js ohjelmistokehyksen kanssa. (Express.js 2016). Express.js voi integroida MongoDB:een ottamalla oikean node.js-liitännäisen käyttöön.

Express.js:n alkuperäinen kehittäjä on TJ Holowaychuk. Kesäkuussa 2014 hallintaoikeudet siirtyivät Amerikkalaiselle yritykselle StrongLoop, joka sulautettiin IBM:ään vuonna 2015. (Express.js 2016). Tammikuussa 2016 IBM ilmoitti, että Express.js siirtyy Node.js Foundationin Incubator-ohjelmaan. (Node.js announcements 2016).

4.4 ANGULAR.JS

4.4.1 YLEISTÄ

Angular.js (tai AngularJS) on pääasiallisesti Googlen ylläpitämä JavaScript-ohjelmistokehys. Sillä on avoin lähdekoodi ja tämän ansiosta on olemassa yksittäisistä henkilöistä ja yhtiöistä koostuva kommuuni, joka auttaa ylläpitämisessä. AngularJS:n yhteisö on myös hyvin aktiivinen ja auttavat toisiaan yksisivuisten sovellusten kehittämisessä. AngularJS:n kehitys alkoi alun perin vuonna 2009 Misko Heveryn toimesta. (AngularJS 2016).

4.4.2 YKSISIVUISET WEB-SOVELLUKSET

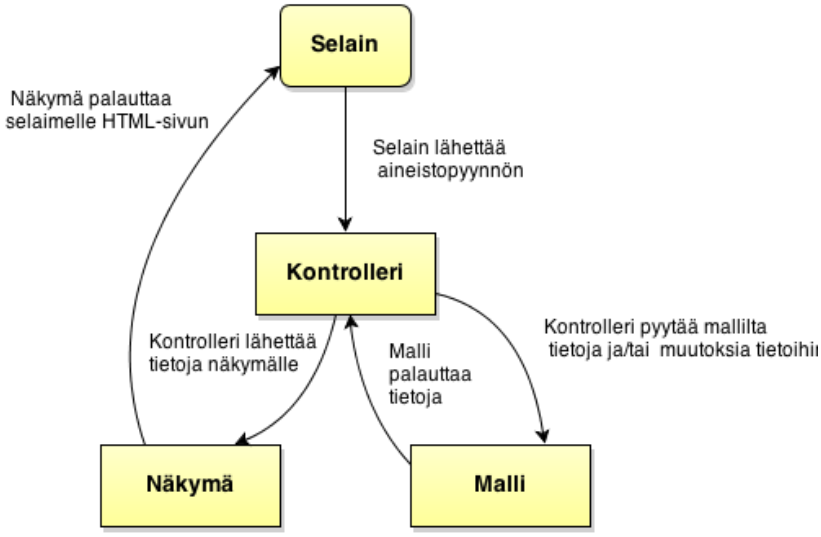
AngularJS:llä tehdään yleensä yksisivuisia web-sovelluksia tai web-sivuja. Tämä tarkoittaa, että sovellus tai sivu simuloi tyypillisiä työpöytä sovelluksia. Yksisivuisissa sovelluksissa joko kaikki tarvittava koodi haetaan kerralla, kun sivu latautuu, tai tarvittavat resurssit ladataan dynaamisesti vastaamaan käyttäjän toimintoja. Mikä erottaa yksisivuiset sovellukset muista selkeimmin on, että sivua ei ladata uudelleen missään välissä, tai kontrolli ei siirry toiselle sivulle. (Single-page application 2016).

AngularJS:n tarkoitus on yksinkertaistaa näiden yksisivuisten sovellusten tekemistä tarjoamalla viitekehyksen MVC-arkkitehtuurille ja muille komponenteille, mitä käytetään ”rikkaissa Internet-sovelluksissa”. Tämä nopeuttaa sovelluksen kehittämistä ja testaamista. (AngularJS 2016).

4.4.3 MVC-ARKKITEHTUURI

MVC-arkkitehtuurissa koodi jaetaan kolmeen osaan: näkymiin, kontrollereihin ja malleihin (engl. views, controllers ja models). Mallit abstrahoivat tietokannasta saatuja tietoja ohjelmointirajapinnaksi muulle koodille käytettäväksi. Näkymät hoitavat html-koodin ja palauttaa selaimelle näytettävän html-sivun. Kontrollerit ottavat vastaan

käyttäjän antamat aineistopyynnöt, hakevat ja päivittävät tietoa malleja hyödyntäen ja välittävät tämän tiedon näkymälle. (Arkkitehtuuri ja MVC n.d.).



KUVIO 5, MVC-MALLIN VASTUUJAKO (ARKKITEHTUURI JA MVC)

4.4.4 ANGULAR.JS:N TOIMINTA

AngularJS:n viitekehys toimii siten, että ensin se lukee HTML-sivulta elementit, joihin on sisällytetty AngularJS:n omia html-määritteitä. AngularJS tulkitsee nämä direktiiveiksi ja lukee näiden kautta, mitä toiminnallisuutta kyseiseen kohtaan halutaan. AngularJS siten yhdistää elementtiin toiminnallisuuden mukaisen syötteen luku- tai kirjoitusoperaation mallipohjan mukaisesti muuttujiin liitettynä. Näitä muuttujia voidaan kontrolloida staattisesti, tai kuten tässä opinnäyteyön projektissa, dynaamisesti lukemalla arvoja tietovarastosta JSONin avulla. (AngularJS 2016).

4.5 NODE.JS

4.5.1 YLEISTÄ

Node.js on kirjasto palvelinpään JavaScript-ajoympäristölle. Perinteisistä sovelluskehyksistä node.js poikkeaa siten, että se on tapahtumavetoinen: sovellukset ohjelmoidaan nodelle lähtökohtaisesti asynkronisesti callback-mekanismien pohjalle. Tämän ansiosta yksittäisiin http-pyyntöihin voidaan kytkeä useita asynkronisia palvelukutsuja. (Salonen 2012.) Tapahtumavetoisena viitekehyksenä node.js on suunniteltu skaalautuville internet sovelluksille.

Node.js kehitti alun perin Ryan Dahl vuonna 2009 ja tuki vain Linuxia. Nykyään node.js on Node.js Foundationin hallitsema. Suuria node.js:n teknologiaa käyttäviä yrityksiä ovat esimerkiksi: GoDaddy, IBM, LinkedIn, Microsoft, Netflix, PayPal, SAP ja Yahoo. (Node.js – Wikipedia 2016).

4.5.2 Säikeistys

Node.js ei käytä lainkaan säikeitä, koska yleensä säie-pohjainen verkkotyöskentely on suhteellisen epätehokas ja hyvin hankala käyttää. Lisäksi node.js:ssä ei ole ollenkaan lukkoja: melkein mikään node.js:n funktio ei suoraan käsittele sisään- tai ulostuloja, joten prosessi ei koskaan lukitu. Koska mikään ei lukitu, on node.js:llä paljon helpompi kehittää skaalautuvia järjestelmiä. Vaikka node.js on suunniteltu ilman säikeiden käyttöä, voi sillä silti hyödyntää useamman ytimen käyttöä: node.js:llä voi luoda lapsiprosesseja ja jakaa ydinten rasitusta jakamalla suoritinkantoja prosessien välillä. (Node.js N.d.).

4.5.3 Tapahtumasilmukat

Node.js on malliltaan samanlainen systeemi, kuin Rubyn Event Machine tai Pythonin Twisted, mutta vie näiden tapahtumamallit hieman pitemmälle: node.js esittää tapahtumasilmukat ohjelmistokielen perustana pelkän koodikirjaston sijaan. Tyypillisesti ohjelmakoodin alussa määritellään halutut toiminnot callback-mekanismeilla ja lopussa

käynnistetään palvelin lukitsevalla kutsulla. Koska node.js:ssä ei tällaisia lukitsevia kutsuja ole, käynnistää se tapahtumasilmukat heti callback-mekanismien jälkeen. Näitä tapahtumasilmukoita ajetaan niin kauan, kunnes callback-pyyntöjä ei enää ole. Tämä on hyvin samantapainen tapa, kuin miten JavaScript toimii selaimessa: tapahtumasilmukat ovat piilotettu käyttäjältä. (Node.js, n.d.).

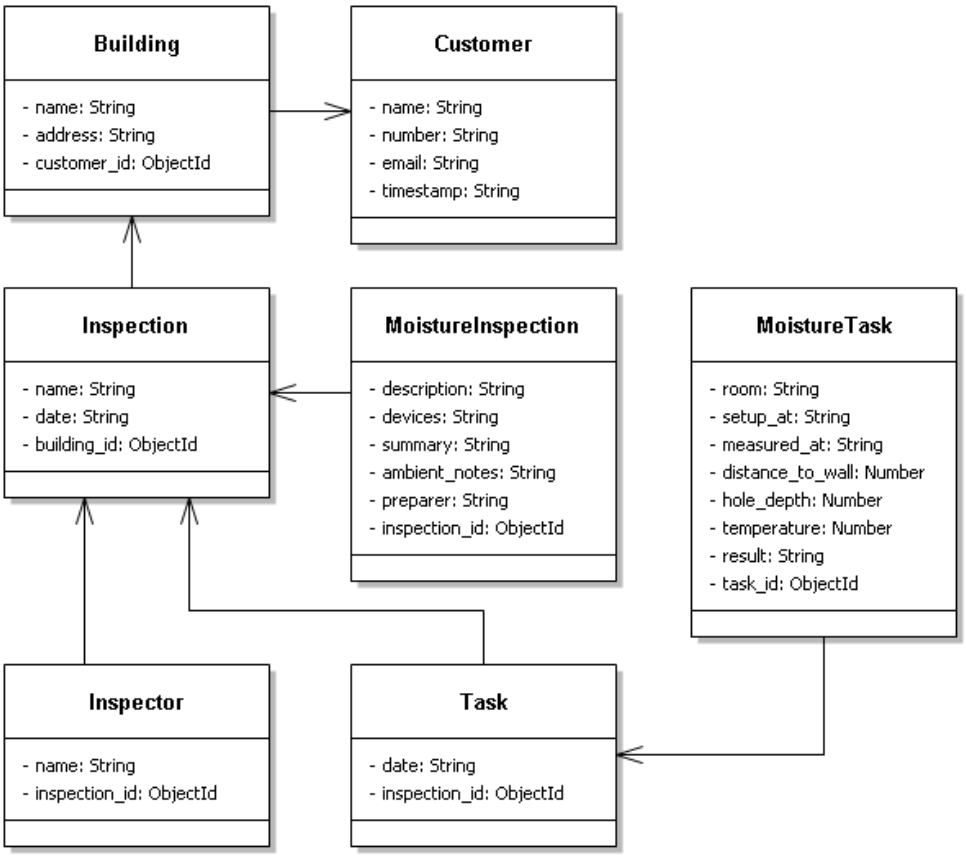
5 SOVELLUKSEN TOTEUTUS

5.1 TIETOKANNAN TOTEUTUS

Koska tietokanta päätettiin tehdä NoSQL-pohjaisella teknologialla, ei tietokantamallin luomiseen tarvittu mitään erillistä ohjelmaa. Tyypillinen lähestymistapa SQL-pohjaiselle olisi rakentaa malli erillisellä ohjelmalla, kuten MySQL Workbench:llä. Tämä vaihe pystyttiin ohittamaan kokonaan ja se säästi paljon aikaa.

Datan muunto SQL-pohjaisesta tietokannasta mobiililaitteesta MongoDB:en onnistuu esimerkiksi työkalulla ”Mongify”, ja tässä projektissa käytetyllä teknologialla nimeltään ”Mongoose”. (Kaplan, M. 2014).

Alla olevassa kaaviossa on havainnollistettu, miten Mongoosella koodin puolella on rakennettu perinteistä tietokantamallia vastaava rakenne MongoDB:lle. Esimerkiksi kosteustarkastustaulun (MoistureTask) lämpötilakenttä (temperature) on rajoitettu sisältämään vain lukuja. Kuviossa on myös kuvattu, kuinka taulut ovat relaatiossa toistensa kanssa: rakennustauluun (Building) voidaan sitoa tietyn asiakkaan (Customer) id. Näin alkaa jo syntyä tyypillisen SQL-tyyppisen tietokannan mukainen relaatiomalli.



KUVIO 6, GRAAFINEN ESITYS MONGOOSELLA KEHITETYSTÄ TIETOKANTAMALLIA VASTAAVASTA RAKENTEESTA

Eräs tärkeä asia, mikä tulee ottaa huomioon tietojen poistamisessa on, poistetaanko kaikki tieto tietokannasta sovelluksen puolelta nappia painamalla. Toinen tapa on merkitä poistetut tiedot ”poistetuksi” tietokantaan, mutta säilytetään tietokannassa. Sovelluksen puolella ei näytetä tällöin tietoja, jotka ovat merkitty poistetuksi tietokannassa. Koska tähän ei ollut mitään vaatimusta, päätettiin, että tiedot ajetaan heti pois tietokannasta, jos ne web-sovelluksen puolelta poistaa.

5.2 WEB-SOVELLUKSEN RAKENTAMINEN

5.2.1 NÄKYMÄT

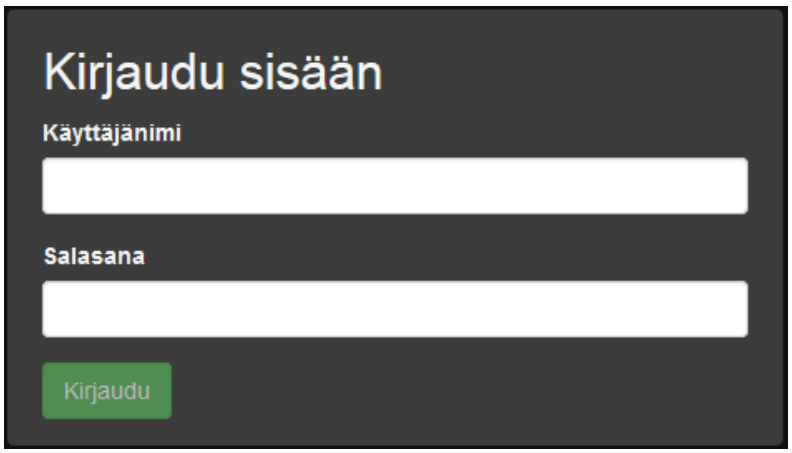
Kappaleessa 2.1.5 mainittiin, kuinka asiakasohjelma koostuu useasta eri näkymästä. Jokaisella näkymällä on tietty rajattu rooli ja tarkoitus. Koska AngularJS erikoistuu yksisivuisten html-sivujen tekoon, oli sovellus helppo jakaa eri osiin näillä näkymillä.

Opinnäytetyön web-sovellus on kuitenkin sekoitus yksisivuisesta sovelluksesta ja tavallisesta web-sivustosta: päänäkymässä tarvittava koodi ladataan dynaamisesti käyttäjän vaatimusten mukaisesti, mutta jos käyttäjä haluaa esimerkiksi muokata asiakastietoja, siirtää sovellus näkymän ja kontrollin toiselle sivulle. Tämä voitaisiin toteuttaa täysin yksisivuisesti siten, että koodi toisesta näkymästä (asiakasmuokkaus) ladataan näkyviin päänäkymän sisälle. Koska vanhoja Internet Explorerin versioita ei tarvinnut tukea, päätetiin sovellukselle valita AngularJS:n versio 1.4.2. Navigoinnin helpottamiseksi sovellus myös käyttää AngularJS-moduulia nimeltään angular-route. Tämä mahdollistaa eri näkymiin siirtymisen url-linkin avulla, ja niihin voi helposti yhdistää tietyt kontrollerit.

Sovelluksen ulkoasu toteutettiin täysin käyttämällä Bootstrap-kirjastoa. Bootstrap vaatii joko jQuery-kirjaston tai erityisiä AngularJS-kirjastoja, jotta kaikki sen ominaisuudet toimivat oikein. Tähän projektiin valittiin jQuery-kirjasto, koska AngularJS-kirjastot eivät tukeneet uusinta Bootstrap-versiota. Pienillä CSS tyylimuutoksilla saatiin aikaan tumma teema sivuille.

5.2.2 Kirjautuminen

Kirjautumisnäkymä koostuu kahdesta input-elementistä: käyttäjänimi ja salasana. Nämä kentät ovat vaadittuja täyttää ja sovellus näyttää virheilmoituksen, jos tiedot puuttuvat tai ovat virheelliset. Näiden input-elementtien lisäksi on kirjaudu-painike ja testauskäyttöön tehty rekisteröidy-painike, jolla voi helposti lisätä käyttäjiä tietokantaan. Rekisteröintipainike on hävitetty julkaisuversiossa (ks. Kuvio 7).



KUVIO 7, KIRJAUTUMISNÄKYMÄ SOVELLUKSESSA

Ohjelmistokoodin puolella käyttäjä näkee vain kirjautumisnäkymän. Kun käyttäjä painaa kirjaudu-painiketta, ohjaa AngularJS komennon kirjautumisnäkymästä kirjautumis-kontrolleriin (login-controller). Kirjautumis-kontrollerissa käyttäjän syöttämät tiedot lähetetään todennus-palveluun (authentication-service), joka tarkastaa käyttämällä käyttäjäpalvelua (user-service), ovatko syötetyt tiedot oikein. Käyttäjäpalvelu luo/tallentaa/hakee tiedot tietokannasta REST-kutsuja käyttämällä.

Salasana käännetään Base64-menetelmällä tietokantaan, jotta se ei olisi missään välissä suoraan luettavissa. Base64-koodausta käytetään, kun halutaan tallentaa binääridataa ja pitää se koskemattomana. Käytännössä se ottaa salasanan ASCII-merkkien numeeriset arvot ja muuttaa ne avainmerkkijonon perusteella Base64-formaattiin. Tässä projektissa käytetty avainmerkkijono koostuu merkeistä A-Z, a-z, 0-9, +, / ja =, koostaen yhdessä vaadittavat 64 merkkiä. Esimerkiksi sana ”Testi” käännettynä Base64:ksi voisi näyttää seuraavalta: ”VGVzdGk=”.

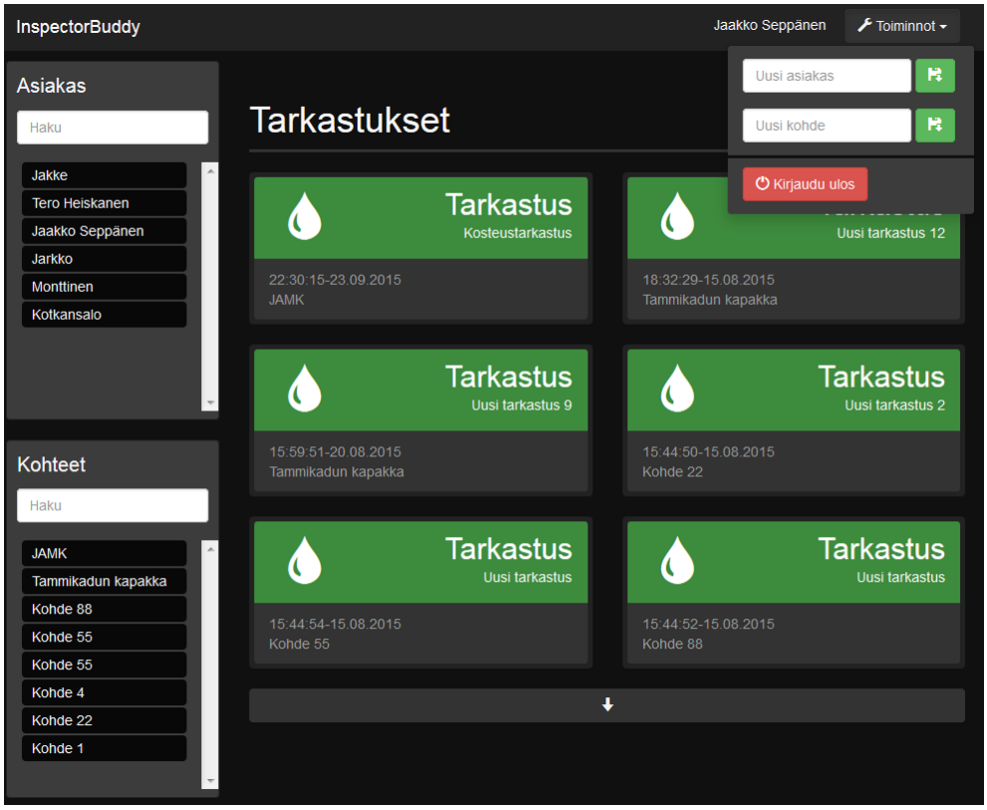
Base64 ei missään nimessä ole itsessään tietoturvallinen tapa säilöä salasanoja: se suojaa salasanat vain paljailta silmiltä, mutta kuka tahansa voi sopivalla ohjelmalla kääntää base64-merkkijonon takaisin luettavaksi tekstiksi. Sillä lähinnä varmistetaankin, että käyttäjänimi ja salasana kuuluvat samaan ”merkkiavaruuteen” eli Base64:ssä käytettyyn avainmerkkijonoon.

5.2.3 PÄÄ-NÄKYMÄ

Onnistuneen kirjautumisen jälkeen käyttäjä ohjataan pää-näkymään. Pää-näkymän tarkoitus on näyttää kaikki tärkeä tieto välittömästi käyttäjälle ja tästä näkymästä käyttäjän on helppo navigoida muihin näkymiin. Pää-näkymä koostuu kolmesta eri alueesta: yläpalkista, vasemmasta sivupalkista ja sisältötilasta.

Yläpalkki näyttää sovelluksen nimen ”InspectorBuddy” ja oikealla reunassa näkyy, kuka on kirjautuneena sisään hallintatyökaluun. Yläpalkissa on myös toiminnot-painike, jota painamalla avautuu muutama perusominaisuus, kuten uuden asiakkaan ja kohteen manuaalinen lisäys ja uloskirjautumispainike (ks. Kuvio 8). Uuden kohteen lisäys on osittain linkitetty jo luotuihin asiakkaisiin: jos käyttäjä on aktivoinut valmiin asiakkaan vasemmasta sivupalkista, voi hän linkittää uuden kohteen tähän asiakkaaseen luomalla sen manuaalisesti yläpalkista. Uusiin luotuihin asiakkaisiin tai kohteisiin ei tule mitään muuta tietoa, kuin nimi ja MongoDB:n oma generoitu id-tunnus.

Vasemmassa sivupalkissa näkyy listana kaikki tietokannan asiakkaat ja kohteet. Niille on rakennettu hakufiltteri, joka suodattaa haut merkkijonon perusteella. Isoilla tai pienillä kirjaimilla haussa ei ole merkitystä. Asiakas- ja kohdelista pystyy näyttämään kerralla 9 yksilöä kerrallaan ja tietyn kohteen/asiakkaan aktivointi tapahtuu yksinkertaisesti sitä klikkaamalla. Kohteet myös suodatetaan aktivoidun asiakkaan mukaan, eli kohdelistassa näkyvät aktivoituun asiakkaaseen linkitetyt kohteet. Suodatus ei tapahdu päinvastoin.



KUVIO 8, PÄÄ-NÄKYMÄ SOVELLUKSESSA

Sisältötila on ylä- ja sivupalkin väliin jäävä tila, missä näytetään käyttäjän haluama tieto. Kun käyttäjä aluksi kirjautuu sisään, näkyy sisältötilassa kuusi uusinta tarkastusta aikaleiman mukaan järjestettynä. Käyttäjä voi halutessaan näyttää lisää tarkastuksia painamalla alaspäin osoittavaa nuolipainiketta tarkastuksien alapuolella (ks. Kuvio 8).

Luodut tarkastukset näkyvät sisältötilassa isoina elementteinä, jotka rakennetaan hyödyntäen AngularJS:n direktiivejä. Direktiiveillä voi helposti tehdä dynaamisesti isoja HTML-kokonaisuuksia, jotka voivat toimia itsenäisesti. Tarkastus-elementeissä näkyy tarkastuksen nimi, luontipäivämäärä, tarkastuksen kohteen nimi ja tarkastuksen tyyppi isona ikonina. Kuviossa 8 näkyy, kuinka kosteustarkastukset ovat kuvattu isolla tippa-ikonilla. Tarkastus-elementtiä painamalla ohjelma avaa tarkastuksenmuokkaus-näkymän.

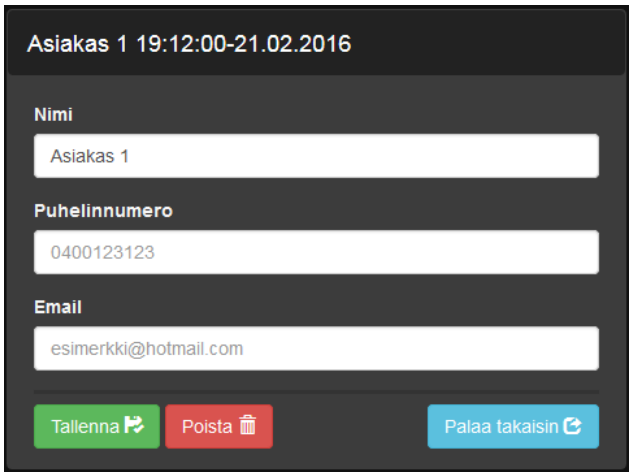
Sisältötila näyttää myös valitun asiakkaan, asiakkaan muokkaus-napin ja asiakkaan aikaleiman. Asiakkaan lisäksi sisältötila pystyy näyttämään valitun kohteen, kohteen

muokkaus-napin ja kaikki kohteeseen linkitetyt tarkastukset. Valittuun kohteeseen pystyy lisäämään uusia tarkastuksia sisältötilasta painamalla vihreää ”+”-nappia.

5.2.4 Asiakasmuokkaus

Asiakasmuokkausnäkymä avautuu, kun käyttäjä on painanut valitun asiakkaan muokkaus-nappia pää-näkymässä. Tämä laukaisee funktion pää-näkymän kontrollerissa, mikä ohjaa AngularJS:n routeProvideria hyväksikäyttäen avaamaan ohjelmassa oikean näkymän ja ottamaan käyttöön oikean kontrollerin tälle näkymälle. Asiakasmuokkauksen kontrolleriin injektoidaan tietokannasta asiakkaan tiedot muokattavaksi.

Asiakasmuokkausnäkymän otsikkona on asiakkaan nimi ja asiakkaan aikaleima, mikä päivittyy aina, kun asiakasta muokataan. Asiakasmuokkauksessa asiakkaan nimen voi vaihtaa ja asiakkaaseen voi liittää puhelinnumeron ja sähköpostiosoitteen (ks. Kuvio 9). Muokatut tiedot päivitetään tietokantaan painamalla vihreää tallenna-nappia. Asiakkaan voi myös poistaa tietokannasta painamalla punaista poista-nappia. Koska ohjelma poistaa asiakkaan suoraan tietokannasta, näyttää ohjelma varoitus-modaalin, jossa käyttäjän pitää varmistaa poisto. Asiakasmuokkausnäkymästä poistutaan takaisin pää-näkymään painamalla sinistä ”Palaa takaisin” -nappia.



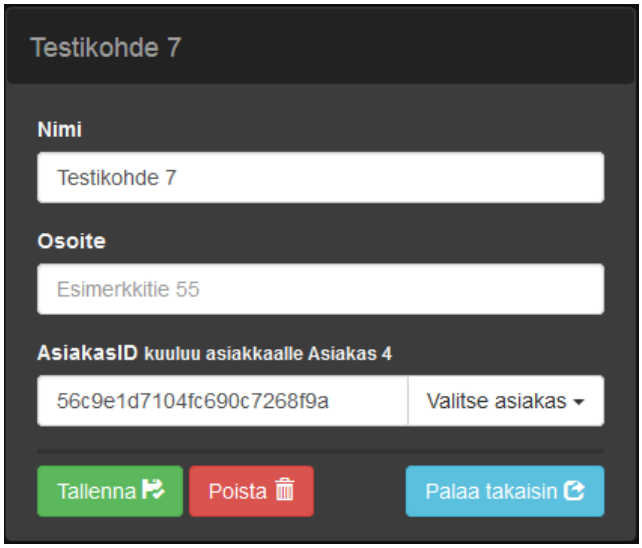
KUVIO 9, ASIAKASMUOKKAUSNÄKYMÄ

Alkuperäisenä suunnitelmana tämä muokkausnäkymä pitäisi näkyä pää-näkymän sisältötilassa, hyväksikäyttäen AngularJS:n yksisivuisuutta. Toinen mahdollisuus olisi ollut näyttää näkymä modaalina. Modaali on uusi ikkuna pää-näkymän päällä, mutta se estää pää-näkymän käyttämisen, ennen kuin modaali on suljettu.

5.2.5 Kohdemuokkaus

Kohdemuokkausnäkymä on toiminnaltaan hyvin samanlainen asiakasmuokkausnäkymän kanssa. Se avataan painamalla aktivoidun kohteen muokkaus-painiketta pää-näkymässä. Muokkaus-painike ajaa pää-näkymän kontrollerissa funktion, joka ohjaa sovelluksen näyttämään oikean html-sivun ja käyttämään oikeaa kontrolleria. Kohdemuokkauksen kontrolleriin pitää injektoida tietokannasta kohteiden lisäksi kaikki asiakkaat, koska muokattava kohde tulee pystyä linkittämään tiettyyn asiakkaaseen.

Ulkoasultaan kohdemuokkausnäkymä on hyvin samanlainen asiakasmuokkausnäkymän kanssa. Uutena ominaisuutena kuitenkin on asiakkaan linkittäminen kohteeseen, mikä tapahtuu valitsemalla haluttu asiakas pudotuslistasta. Koska tiedot tallennetaan suoraan tekstikentistä tietokantaan, näyttää se käyttäjälle asiakkaan id:n, mihin kohde linkitetään. Näkymä kuitenkin näyttää id:seen kuuluvan asiakkaan nimen, jotta käyttäjä pystyy tunnistamaan sen helposti (ks. Kuvio 10).



KUVIO 10, KOHDEMUOKKAUSNÄKYMÄ

Tiedot tallennetaan tietokantaan vihreää tallenna-painiketta painettaessa. Kohdemuokkauksen kontrolleri päivittää sekä kohteen, että kohteelle valitun asiakkaan. Asiakkaan päivitys on tärkeää, koska sen aikaleima pitää päivittää aina, kun siihen linkitettyjä kohteita päivittää. Kohteen poisto tietokannasta tapahtuu painamalla punaista poista-nappia. Käyttäjälle näytetään taas modaali-ikkuna, jossa käyttäjän pitää varmistaa kohteen poisto, jotta ei vahingollisia poistoja tapahtuisi. Kohteen muokkausnäkymästä palataan takaisin painamalla sinistä ”Palaa takaisin” -painiketta.

5.2.6 TARKASTUSMUOKKAUS

Tarkastusmuokkausnäkymään pääsee pää-näkymästä painamalla tarkastus-elementtiä. Koska tarkastus-elementit ovat direktiivejä, niillä on oma kontrolleri, joka ohjaa oikealle muokkaussivulle. Tarkastusmuokkausnäkymä on taas hyvin samankaltainen asiakkaan- ja kohteenmuokkausnäkymän kanssa. Tarkastusmuokkausnäkymässä voi muokata tarkastuksen perustietoja, kuten vaihtaa tarkastuksen nimen, linkittää tarkastajan pudotusvalikosta tai muokata tarkastuksen päivämäärää.

Näiden perustietojen lisäksi tarkastusmuokkauksessa on toinen sivuikkuna perustietojen lisäksi, jonka näytettävät tiedot riippuvat siitä, onko tarkastus kosteustarkastus vai

rakennustarkastus. Näitä tietoja voidaan muokata samaan tapaan muiden muokkausnäkymien tavalla ja ne tallentuvat tietokantaan vihreää tallenna-nappia painaessa. PDF:n luontinäkymään siirrytään ”Luo PDF” -painikkeesta. Tarkastuksen voi poistaa tietokannasta poista-painikkeella ja tarkastuksen muokkauksesta pääsee takaisin pää-näkymään ”Palaa takaisin” -painikkeella.

5.2.7 PDF-NÄKYMÄ

PDF-näkymän voi avata sen tarkastuksen muokkausnäkymästä, mistä PDF-tuloste halutaan tehdä. Näkymä sisältää syöte-kentät valmiiksi täytettyinä tarkastuksen tiedoista, mutta niitä voi myös muokata näkymässä. Näkymä sisältää ”Näytä PDF” -painikkeen, mikä näyttää käyttäjälle lopullisen, tulostettavan PDF:n.

5.3 PALVELIN

Palvelin kehitettiin Node.js:n versio 4.4.0:lla koska se on vakaa ja toimii myös tarvittaessa useimmilla UNIX-käyttöjärjestelmillä. Node.js:n kaverina toimii Express.js mikä hoitaa useimmat palvelimen asetukset ja toiminnot automaattisesti. Mongoose hoitaa toiminnot palvelimen ja tietokannan välillä käyttämällä url-linkkejä. Esimerkiksi osoite voisi olla ”localhost:3000/customers”, mikä palauttaa kaikki tietokannan asiakkaat JSON-muodossa, mikä taasen on helppo käsitellä AngularJS:llä.

6 TULOKSET

6.1 TESTAUS

Vaatimusmäärittelyn mukaan sovellus testataan vasta, kun web-sovellus ja mobiilisovellus molemmat ovat valmiita. Koska mobiilisovellusta ei kehitetty samaan aikaan web-sovelluksen kanssa, lopullista testausta käytännön tilanteessa ei ole tehty.

Testaaminen tapahtuu sekä projektin kehittäjien, että PJR-Team työntekijöiden toimesta. Kehittäjät testaavat, että kaikki ominaisuudet toimivat oikein. Projektin tilaaja tarkastaa, että ominaisuudet toimivat vaatimusten mukaisesti. Erillisiä testitapauksia ei kirjattu eikä sovellukseen ohjelmoitu mitään testejä.

AngularJS:llä yksikkötestaus tapahtuu helposti hyvin suositulla Jasmine - nimisellä viitekehyksellä. Se tarjoaa testaamiseen tarkoitettuja funktioita ja pitää testit hyvin dokumentoituina ja erillisinä isoissakin projekteissa. (Unit Testing 2016)

6.2 WEB-SOVELLUS

Web-sovelluksesta valmistui funktionaalinen hallintatyökalu, joka pystyy lataamaan tietoja tietokannasta muokattavaksi ja jolla voi lisätä tietoja tietokantaan. Se on suojattu yksinkertaisella kirjautumisnäkymällä. Web-sovelluksesta jäi uupumaan PDF-dokumentin luontiominaisuus, koska sitä ei ehditty suunnitella ja kehittää määrätyssä ajassa.

Ongelmia tuli muun muassa siinä, kun yritettiin näyttää kohteenmuokkausnäkymässä asiakkaan nimeä id:n perusteella. Koska kohteenmuokkausnäkymään pitää ladata kaikki asiakkaat ja kohteet tietokannasta sen avautuessa, ei id:lle keretä samanaikaisesti selvittämään asiakkaan nimeä. Tämä ratkaistiin siten, että tietojen lataaminen viedään niin sanottujen ”lupausten” kautta: ohjelma ei aja seuraavia funktioita ennen kuin kaikki tiedot tietokannasta on ladattu. Kun tiedot tietokannoista on varmasti ladattu, palauttaa lupaus oikeuden jatkaa funktioiden suorittamista, jolloin id:lle kuuluvan nimen voi turvallistesti tunnistaa.

6.3 JÄRJESTELMÄ

Koko järjestelmästä valmistui vain suurin osa web-sovelluksesta. Tähän myös kuuluu tietokannan luonti palvelimelle. Sovellusta ei kuitenkaan missään välissä asennettu yrityksen palvelimelle. Mobiiliosuuden kehitystä ei aloitettu, joten järjestelmän toimivuutta mobiiliosan ja web-sovelluksen välillä ei voitu testata.

Teoriassa järjestelmä toimisi, koska mobiiliosuus ja web-sovellus käyttävät samaa tietokantaa. Jos sinne pystyy mobiililaitteella laittamaan tietoja oikeassa muodossa, web-sovellus kykenee tulkitsemaan ne.

7 JOHTOPÄÄTÖKSET

7.1 ONNISTUMISET

MEAN-sovelluskokoelma osoittautui loistavaksi valinnaksi web-sovelluksia kehitettäessä. Nämä neljä teknologiaa on melko nopea oppia, vaikka ei aluksi niistä tietäisikään mitään. Vaikein näistä teknologioista oppia on AngularJS, mutta tästä projektitiimillä oli jo hieman kokemusta. Sovelluksen kehitys MEAN-sovelluskokoelmalla tuntui varsin nopealta ja harvemmin tuli ongelmia tai esteitä itse teknologioiden kanssa. Tämän ansiosta pystyttiin enemmän keskittymään puhtaasti koodin puolelta ongelmien ratkaisuun.

Sovelluskokoelma kattaa myös täysin palvelimen ja asiakaspäätteen kehittämisen tietokantoineen, joten kokemuksen kannalta projekti oli erittäin rikas ja monipuolinen. Koska projektia alettiin alun perin tehdä myös kahden henkilön voimin, tuli suunnitteluvaiheesta tärkeämpi. Tämä myös vaati sen, että järjestelmällinen

kehitysympäristö tuli pystyttää ja työtehtävät tuli jakaa. Kaikki tämä onnistui tehokkaasti, aikataulutuksia lukuun ottamatta.

Web-sovelluksesta tuli myös ulkoisesti siistin näköinen. Bootstrapin valmiita CSS-tyylejä ei tarvinnut paljoa muokata, jotta sovellus täytti vaatimusten mukaisen ulkoasun. Vaikka skaalautuvuus ei ollut mikään alkuperäinen tavoite, tuli sovelluksesta hyvin skaalautuva minimissään 768 pikseliin asti (selaimen leveydeltä). Tämän jälkeen html-komponentit alkavat hävitä sivujen ulkopuolelle.

7.2 EPÄONNISTUMISET

Suurin ongelma projektia kehitettäessä oli aikataulutus. Projekti oli liian suuri sille määrätylle aikataululle. Vaikka sovelluskokoelmalla kehitys tapahtui hyvin nopeasti, sen käyttö piti ensin opetella ja tämä vaati hyvin paljon aikaa, mitä ei alun perin otettu tarpeeksi huomioon. Jos projekti tehtäisiin uudestaan, aikataulutus olisi ollut paljon tarkempi.

Hämäännys siitä, miten tietokannat pitäisi sulauttaa yhteen mobiilin ja palvelimen välillä kehityksen aikana, aiheutti ajanhukkaa. Tämä olisi pitänyt selvittää suunnitteluvaiheessa täydelliseksi ja ilman oletuksia. Kehitysvaiheessa vaihdettiin tapoja monta kertaa, miten tämä sulautus käytännössä tapahtuisi, vaikka mobiiliosan kehitystä ei edes tehty samaan aikaan. Sekaannus aiheutui siitä, että aina valittu ratkaisu mobiilin ja palvelimen välille ei myöhemmin tulekaan toimimaan erinäisistä syistä.

Koska mobiiliosaa ei kehitetty samaan aikaan, koko systeemiä ei päästy kokeilemaan yhdessä missään välissä. Vaikka web-sovellus ei olekaan ominaisuuksiltaan täydellinen, olisi sitä voinut jo testata mobiilisovelluksen kanssa. Näin oltaisiin huomattu, miten se käytännössä toimii ja onko siitä edes mitään hyötyä sellaisenaan ja olisiko jotain ominaisuuksia pitänyt kehittää toisin.

7.3 KEHITYSEHDOTUKSET

Web-sovelluksen ollessa vain osittain valmis kehitysehdotuksia on paljon. Edellisen kappaleen mukaan sitä pitäisi testata yhdessä mobiilisovelluksen kanssa, jotta voitaisiin selvittää, tarvitseeko jotain ominaisuuksia vaihtaa. Kosteustarkastusten lisäksi pitäisi lisätä mahdollisuus myös tehdä kuntotarkastuksia kohteille. Sovelluksella olisi myös hyvä pystyä tarkastelemaan otettuja kuva- ja ääninäytteitä.

Mobiilipuoli suunniteltiin toimimaan vain tietyllä mobiililaitteella, mitä yritys PJR-Team käyttäisi. Tätä voitaisiin parantaa tekemällä mobiilisovelluksesta toimiva yleisimmillä mobiililaitteilla. Tällaiselle ratkaisulle tulisi suunnitella uusi systeemi, miten synkronointi mobiililaitteen ja tietokannan välillä tapahtuisi.

LÄHTEET

AngularJS. 2016. Wikipedia – vapaa tietosanakirja. Viitattu 25.2.2016. https://en.wikipedia.org/wiki/AngularJS

Arkkitehtuuri ja MVC. N.d. Advanced Kittenry – Tietokantasovellusohjeet. Viitattu 25.2.2016. https://advancedkittenry.github.io/koodaaminen/arkkitehtuuri/index.html

Bitbucket. 2016. Wikipedia – vapaa tietosanakirja. Viitattu 24.2.2016. https://en.wikipedia.org/wiki/Bitbucket

Buecheler, C. 2015. Creating a simple restful web app with Node.js, Express, and MongoDB. Viitattu 22.9.2015. http://cwbuecheler.com/web/tutorials/2014/restful-web-app-node-express-mongodb/

Confluence. 2015. Features. Viitattu 16.9.2015. https://www.atlassian.com/software/confluence

Data Modeling Introduction. N.d. Viitattu 16.9.2015. http://docs.mongodb.org/manual/core/data-modeling-introduction/?_ga=1.16437686.1455102840.1442395651

Jira (ohjelmisto). 2016. Wikipedia – vapaa tietosanakirja. Viitattu 22.2.2016. https://fi.wikipedia.org/wiki/Jira_(ohjelmisto)

JIRA Software. 2015. Atlassian Shop. Viitattu 22.2.2016. https://www.atlassian.com/purchase/product/jira-software

Karpov, V. 2014. The MEAN Stack: MongoDB, ExpressJS, AngularJS and Node.js. Viitattu 18.2.2016. http://blog.mongodb.org/post/49262866911/the-mean-stack-mongodb-expressjs-angularjs-and

Kaplan, M. 2014. How to Migrate from MySQL to MongoDB. Viitattu 19.2.2016. https://dzone.com/articles/how-migrate-mysql-mongodb

Laine, H. 2005. Tietokantojen perusteet. Viitattu 19.2.2016. http://www.cs.helsinki.fi/u/laine/tkp/tietomallit/kasiteanalyysi.html

MongoDB. N.d. How does MongoDB work? Viitattu 25.2.2016. https://www.mongodb.com/what-is-mongodb

Node.js. N.d. Viitattu 25.5.2016. https://nodejs.org/en/about/

PJR-Team. N.d. Viitattu 15.9.2015. http://pjr-team.fi/index_yrityksesta.php

Salonen, J. 2012. Johdanto JavaScript-sovellusten kehitykseen Node.js:llä. Viitattu 16.9.2015. http://blite.iki.fi/artikkelit/javascript-nodejs-johdanto/

Single-page application. 2016. Wikipedia – vapaa tietosanakirja. Viitattu 25.2.2016. https://en.wikipedia.org/wiki/Single-page_application

SourceTree. 2016. Overview. Viitattu 24.2.2016. https://www.atlassian.com/software/sourcetree/overview/

Unit Testing. N.d. Additional tools for testing Angular applications. Viitattu 23.3.2016 https://docs.angularjs.org/guide/unit-testing

What is JavaScript? 2015. About JavaScript. Viitattu 23.3.2016. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

What is jQuery?, N.d. Viitattu 23.2.2016. https://jquery.com/

What is npm?, N.d. Viitattu 22.2.2016. https://docs.npmjs.com/getting-started/what-is-npm